Bigand Nicolas
N4



Projet Technique N4 - AI²R
Coupe de France de Robotique 2004
"Coconut Rugby"

Partie VII :
Intelligence Artificielle

Sommaire

I) Les composants du robot

En début d'année, nous avons fait un cahier des charges pour que le robot puisse être efficace à la coupe de France de Robotique. Notre souhait était de faire le robot le plus simple possible et le plus performant. Pour cela nous avons décider de faire des parties indépendantes, c'est à dire qui ne demande pas un pilotage constant. Nous voulions des parties qui donnent les informations quand elles sont sollicitées mais qui sont indépendantes le reste du temps.

Nous avons donc séparé le robot en 5 grandes parties :
- Le déplacement
- La caméra
- Les balises
- Les capteurs d'évitement
- La gestion des moteurs de stockage et de tir

Chaque partie ne nécessite un pilotage du cerveau du robot uniquement lorsqu'il a besoin d'information. Ainsi nous avons créer un robot qui ne demandent que très peu de puissance de calcul.
Nous avons donc choisi de prendre un ST7 pour piloter l'ensemble du robot.
Ce microcontroleur a l'avantage de posséder 6 ports d'entrée – sortie de 8 bits ce qui permet de pouvoir communiquer avec un grand nombre de carte électronique à condition que celles-ci ne demande pas trop d'I/O :

II) Le ST7

C'est un microcontroleur 8 bits de chez STMicroelectronics
Il est cadencé à 8Mhz et possède 48 I/Os.
Il possède 32 ko de Rom et 256 octects de RAM
L'école possède des plaquettes de développement qui facilite grandement le programmation.
Nous utilisons un Compilateur C, COSMIC, qui nous permet de faire tous nos code en C. CE compilateur convertit ensuite le code en assembleur. A noter qu'il est toujours possible d'utiliser des instructions assembleurs dans le code en C (exemple : JUMP, NOP).
Ce microcontroleur est très pratique pour piloter des interfaces, des moteurs. Par contre, il est assez lent ce qui exclut les calculs trigonométriques et les calculs de traitements d'image. Il faut alors trouver des techniques de codage pour palier à se problème. Si le processeur n'est pas assez intelligent, c'est au programmeur de l'être… ;-)
Nous l'avons exploiter au maximum cette année (RAM saturé, programme de 28ko..) mais nous avons put nous rendre compte qu'il était tout à fait possible de piloter tout un robot via un simple microcontroleur. Par contre cela demandait des concessions au niveau du codage et au niveau des cartes électroniques.
C'est pour cela que nous avions besoin de cartes électroniques indépendantes.

Nous avons donc créer une carte mère dont le processeur est un ST7 et qui pilote l'ensemble du robot.

Les connections aux différentes cartes se faisaient via des connecteurs HE10-10 ou HE10-20 (pour l'asservissement) et via des borniers.

Cette carte a été très efficace et très fiable.

Répartition des ports du ST7 :

Port A : connection Carte Capteurs :
- A0 : Interruption (contact avec un obstacle)
- A1 : Bumpers avant gauche
- A2 : Bumpers avant droit
- A3 : Bumpers droite
- A4 : Bumpers arrière droite
- A5 : Bumpers arrière
- A6 : Bumpers arrière gauche
- A7 : Bumpers gauche

Port B : Bus de données pour la carte HCTL1100

Port C : Carte Capteurs : 7 bit pour les capteurs IR
- C1 : Capteur gauche
- C2 : Capteur avant gauche
- C3 : Capteur avant droite
- C4 : Capteur droite
- C5 : Capteur coté droit
- C6 : Capteur arrière
- C7 : Capteur coté gauche

Port D : Carte HCTL1100 : 8 bits de pilotage des puces
- D0 = OE_gauche
- D1 = SYNC
- D2 = CS_droit
- D3 = CS_gauche
- D4 = ALE
- D5 = R_W
- D6 = RESET
- D7 = OE_droit

Port E :
- E0 : Caméra : Port Série
- E1 : Caméra : Port Série
- E5 : Balise : Détection d'une balise
- E6 : Balise : sortie moteur pas à pas

Port F : Pilotage des moteurs de tir
- F0 : interuption de fin de stockage de balle
- F1 : moteur des rouleaux
- F4 : moteur tapis

III) Optimisation du code

Suite à l'intégration de la caméra dans le même ST7 que l'asservissement, nous avons été confronté à un problème de taille de RAM. En effet, la RAM du ST7 est limité est 256 octects. La RAM est principalement utilisée pour stocker les variables.
Ce problème nous est apparu lors du Debuggage avec l'émulateur ST7, nous avions des accès mémoires interdits. Nous en avons déduit que la RAM était saturée et qu'il fallait optimiser notre code. Ainsi, nous avons décidé de n'utiliser que des variables globales pour tous les programmes.
Nous avons donc optimiser le code du déplacement afin de réduire le nombre de variable utilisé.

Nous avons enlevé tous les paramètres des fonctions car cela prenait des variables en RAM, nous avons enlevé toutes les variables locales qui prenaient aussi de la place en RAM. Pour les paramètres des fonctions nous utilisons les variables para1,2,3,4 qui stockent les paramètres d'entrées et qui peuvent être réutilisés à l'intérieur de la fonction.
Ce code est optimisé au maximum mais le code a perdu en visibilité :

Ex : écriture d'une valeur dans un registre
para1 = adresse du registre ; para2 = valeur; write_hctl_gauche();
à la place de
void write_hctl_gauche(unsigned char adresse,unsigned char donne)

Finalement nous utilisions uniquement ces variables externes pour toutes les fonctions :

```
extern unsigned char tableau_mvt[21];
extern unsigned int boucle;
extern unsigned int sortie;
extern unsigned int ceinture;
extern int para1;
extern int para2;
extern int para3;
extern int para4;
extern long paralong;
extern unsigned int mot[6];
extern unsigned int mot1[6];
extern unsigned int ref[6];
extern float b;
extern float a;
extern int compteur_mvt;
```

III) Mode de fonctionnement du robot

Nous avons choisi de faire fonctionner le robot de manière séquentielle, c'est à dire qu'il ne fait qu'une action à la fois.

Notre programme est assez compliqué et afin que vous puissiez le comprendre le plus rapidement possible, je vais vous l'expliquer en commençant par vous montrez comment sont effectués les opérations les plus simples puis au fur et à mesure, je complexifierai le code pour intégrer au final toutes les composantes du robot.

a) Commande du déplacement

Le tableau tableau_mvt[21] est au cœur de l'intelligence du robot, c'est dans celui que sont répertoriés les actions de déplacement à effectuer.
Les informations sont par pair dans ce tableau : il faut connaître tableau[i] et tableau[i+1], i : entier pour connaître une action.

En effet les valeurs pairs du tableau (tableau_mvt[0], tableau_mvt[2], tableau_mvt[4]….) correspondent aux actions à effectuer avec le codage suivant:
tableau[i] = '1' → rotation gauche
tableau[i] = '1' → rotation droite
tableau[i] = '1' → avancer
tableau[i] = '1' → reculer

Ensuite les valeurs impairs du tableau correspondent à la valeur associé à l'action, c'est à dire une distance en centimètre si l'action est avancer ou reculer, ou un angle en degré si l'action est rotation gauche ou rotation droite.

Ex :
tableau[0] = '2'
tableau[1] = 90
→ rotation à droite de 90 °

Ensuite si on veut enchaîner des actions il suffit de remplir le tableau 2 par 2 :
tableau[0] = '1'
tableau[1] = 90
tableau[0] = '3'
tableau[1] = 30
tableau[0] = '2'
tableau[1] = 90

Le robot fera une rotation de 90° a gauche, puis avancera de 30 cm puis tournera de 90 ° à gauche.

Le code est alors celui- ci :

```
void cherche_balle(void)  // cette fonction donne les ordres de déplacement
{ tableau_mvt[0] = '3';
tableau_mvt[1] = 30;
tableau_mvt[2] = '4';
tableau_mvt[3] = 30;
tableau_mvt[4] = '0';
}

void mouvement (void)
{
 // fonction qui lit le tableau et fait l'action décrit dans le tableau
 // a la position compteur_mvt
 if (tableau_mvt[compteur_mvt] == '1')
        {para3 =  tableau_mvt[compteur_mvt+1];           rotation();
         para1 = tableau_mvt[compteur_mvt+1]+60 ;        tempo2();     }
if (tableau_mvt[compteur_mvt] == '2')
         { para3 =  tableau_mvt[compteur_mvt+1];          rotation();
           para1 = tableau_mvt[compteur_mvt+1]+60 ;     tempo2(); }
if (tableau_mvt[compteur_mvt] == '3')
         {   para3 =  tableau_mvt[compteur_mvt+1];        avance2();
             para1 = 200;                                 tempo2();          }
if (tableau_mvt[compteur_mvt] == '4')
         { para3 =  tableau_mvt[compteur_mvt+1];          avance2();
            para1 = 150;                                  tempo2(); }
//para3 est le paramètre d'entrée des fonctions avance2 et rotation, elle stocke la
//valeur du déplacement ( distance ou angle )
}

void ia (void)
{
  cherche_balle();          // on définit les actions à effectuer
  compteur_mvt = 0;         // on initialise le pointeur
  while ( tableau_mvt[compteur_mvt] !='0')     // on boucle tant que on ne trouve
        { mouvement ();       // pas '0' correspondant à aucune
           compteur_mvt = compteur_mvt +2;   // action
         }
}
```

b) Intégration de la caméra et des interruptions

On rajoute alors la fonction :
Ramene () : qui va remplir le tableau de mouvement avec l'action à effectuer pour prendre la balle. Cette fonction remplace cherche_balle().
Cette fonction fait tourner le robot de 50 degré lorsqu'il ne trouve pas de balle.
Ex: rotation gauche 30 puis avance 45

@interrupt void ext0_it(void)
C'est la fonction qui est appelé lorsqu'une interruption apparaît sur ei0 (voir datasheet ST7). Elle arrête le déplacement en cours et implémente des commandes d'évitements. J'autorise les interruptions uniquement lors des déplacements c'est à dire que le programme interruption ne peut être lancé uniquement lors d'un déplacement. Nous avons fais cela en supposant que l'on avait plus de change de prendre un obstacle en se déplaçant qu'a l'arrêt.

On implémente dans mouvement les instructions _asm("SIM") et _asm("RIM") qui respectivement bloque et autorise les interruptions: On suppose que le déplacement se déroule pendant la temporisation, on autorise donc les interruptions pendant ce temps là.

```
void mouvement (void)
{ if (tableau_mvt[compteur_mvt] == '1')
        {para3 =  tableau_mvt[compteur_mvt+1];          rotation();
        _asm("RIM");
        para1 = tableau_mvt[compteur_mvt+1]+60 ;  tempo2();_asm("SIM");   }
if (tableau_mvt[compteur_mvt] == '2')
         { para3 =  tableau_mvt[compteur_mvt+1];          rotation();
           _asm("RIM");
           para1 = tableau_mvt[compteur_mvt+1]+60 ; tempo2();_asm("SIM"); }
if (tableau_mvt[compteur_mvt] == '3')
         {   para3 =  tableau_mvt[compteur_mvt+1];        avance2();
            _asm("RIM");
             para1 = 200;                               tempo2();_asm("SIM");}
if (tableau_mvt[compteur_mvt] == '4')
         { para3 =  tableau_mvt[compteur_mvt+1];          avance2();
           _asm("RIM");
            para1 = 150;                                tempo2();_asm("SIM");}
}
```

```
@interrupt void ext0_it(void)
{  _asm("SIM");    // bloque les interruptions afin d'éviter les rebonds
   para1 = PADR;  // stocke la valeur des sorties des bumpers s
                          // sortie bumpers = 0 → Contact
 if ( (para1 & 0x02) == 0x00)     // bumper 2 devant G
     {  init_hctl_gauche_droit ();          //stop du robot ( reset des HCTL)
         tableau_mvt[0] = '4';
         tableau_mvt[1] = 10; // recule 10
         tableau_mvt[2] = '2';
         tableau_mvt[3] = 90; // rotation droite 90 °
         tableau_mvt[4] = '3';
         tableau_mvt[5] = 35; // avance  35
         tableau_mvt[6] = '1';
         tableau_mvt[7] = 90; // rotation gauche 90°
        tableau_mvt[8] = '0';
         paralong = 99999;      // permet de sortir de la boucle de temporisation

     }
para1 = PADR;
if ( (para1 & 0x04) == 0x00) // bumper 3 devant D
     {
                  init_hctl_gauche_droit ();
                  tableau_mvt[0] = '4';
                  tableau_mvt[1] = 20;
                  tableau_mvt[2] = '1';
                  tableau_mvt[3] = 45;
                  tableau_mvt[4] = '3';
                  tableau_mvt[5] = 42;
         tableau_mvt[6] = '2';
         tableau_mvt[7] = 45;
         tableau_mvt[8] = '3';
         tableau_mvt[9] = 42;
         tableau_mvt[10] = '0';
         compteur_mvt = 0;
         paralong = 99999;
         }
         …..
         // Tous les autres cas suivent, pour les voir tous allez en annexes voir le
code d'interruption
}
```

c) Intégration des capteurs IR, de la boucle de fonctionnement

On rajoute alors la fonction :

Check_capteur();
Cette fonction vérifie les sorties des capteurs IR pendant le mouvement. Si ces mouvement correspondent à un cas prédéfini, alors cette fonction arrête le mouvement, et rentre des consignes d'évitements.
Cette fonction remplace la fonction tempo2() et c'est elle qui fera office de temporisation. Elle bouclera jusqu'à ce qu'un obstacle soit détecté ou alors que le mouvement soit terminé.

```
void check_capteur(void)
{
    boucle = PCDR;
    ceinture = boucle & 0x0F;    // Filtre pour les capteurs C1 C2 C3 C4
    temp1 =  boucle & 0x10;      // capteur 5
    temp2 = boucle & 0x20;       // capteur 6
    temp3 =  boucle & 0x40;      // capteur 7

if (ceinture == 0x01) // cas 8 C1 actif
        { paralong = 99999; // permet de sortir de la temporisation
        arret_urgence();             // fonction qui fait arrêter le robot en urgence
         if (temp1 == 0x00 )         // si C5 non actif
                { tableau_mvt[0] = '2';
                  tableau_mvt[1] = 33;
                tableau_mvt[2] = '3';
                tableau_mvt[3] = 18;
                tableau_mvt[4] = '1';
                tableau_mvt[5] = 33;
                tableau_mvt[6] = '3';
                tableau_mvt[7] = 10;
                tableau_mvt[8] = '0';
              compteur_mvt = -2;

               }
          else
                { if ( temp3 == 0x00)      // si C7 non actif C5 actif
                        { tableau_mvt[0] = '1';   // rot G
                          tableau_mvt[1] = 90;
                          tableau_mvt[2] = '0';
                                compteur_mvt = -2;
```

```
        }
      else
        { if ( temp2 == 0x00)      // si C6 non actif C5 et C7 actif
            { tableau_mvt[0] = 1;   // rot G
              tableau_mvt[1] = 180;
              tableau_mvt[2] = '0';
              compteur_mvt = -2;

            }
         else   // C5 C6 et C7 actif
           {tableau_mvt[0] = 1;
              tableau_mvt[1] = 180;
              tableau_mvt[2] = '0';
            compteur_mvt = -2;
              }
        }
      }
  }
….
// ainsi de suite pour les autres cas
```

on remplace ce bout de programme dans mouvement() à la place de tempo2() :

```
    while (paralong < 10*tableau_mvt[compteur_mvt+1])
         {check_capteur ();
          paralong++;
          }
```

Dans ia() on intègre l'initialisation, la boucle de fonctionnement
Le robot cherchera des balles, se dirigera dessus lorsqu'il en verra une et pourra éviter les obstacles

```
void ia (void)
    {
_asm("SIM");
initialisation_port_st7_hctl();
initst7_seul();
initcam();
while(1)
        {
                clear_tableau();
                ramene();
                compteur_mvt = 0;
                while ( tableau_mvt[compteur_mvt] !=0)
                        { mouvement ( compteur_mvt);
                          compteur_mvt = compteur_mvt +2;
                        }
        }
}
```

Vous trouverez cette fonction ia() dans le l'annexe ou se trouve la version finale du code du robot. Vous trouvez aussi d'autre fonctions ia() que je vous ai mises pour vous faciliter l'apprentissage et vos premiers tests.

V) Mode d'emploi pour réutiliser le robot :

Le robot tel que vous le trouverez ne sera pas à 100% opérationnel, je vais donc vous aidez à le faire marcher :
Parties à 100% opérationnelles :
- Caméra
- Asservissement
- Capteurs
- Laser (s'il est branché)

La partie gestion des moteurs est à 10% opérationnel. Lors des derniers tests un des capteurs était reliés sur le bit F0 pour pouvoir détecter une balle via le ST7. La commande des moteurs marche, il faut lancer le programme shoot qui lance la procédure de tir.

Pour commencez, familiarisez vous avec le code final du robot afin de le comprendre dans la globalité. Ensuite vérifier les branchements, la batterie… Ensuite cherchez nicolas Bigand, benjamin bounous ou fred valloti pour vous aider à faire marcher le robot.

Si on est introuvable, bon courage… et suivez ces conseils :

1. Faites votre programme
2.Compiler votre programme
3. Programmer le ST7
4. Go!

Pour compiler n'oubliez pas de selectionner COSMIC comme C builder dans les option de Visual debug

Pour programmer le ST7 , 2 methodes :
- à l'ancienne : vous enlevez le ST7 très très délicatement avec un tournevis ou un outil mais pas avec les doigts (des ST7 s'en souviennent, ils ont des pattes en moins maintenant…), vous la mettez sur la plaquette ST7 et vous reprogrammez le ST7 via le logiciel visual programmer en sélectionnnant socket dans les options
- Via l'ISP : c'est un port spécial qui permet de reprogrammer le ST7 directement sur la carte mère. Pour cela utilisez un cable HE10 et branchez vous sur l'emplacement ISP sur la carte mère, il faut que la plaquette et la carte mère soit sous tension. Ensuite lancez visual programmer, selectionné ISP dans les options et lancé la programmation.
  Notes: L'ISP ne marche pas sur toutes les plaquettes ST7…

Si vous voulez tester l'asservissement, remplacez dans ia() ramene() par cherche_balle() et mettez dans cherche_balle() le déplacement que vous souhaitez.

Si vous remettez ramene() et vous pourrez faire les tests caméra et asservissement.

Si vous voulez travailler sans les interruptions, remplacez dans ia mouvement par mouvement2

VII) Conclusion

Pour conclure, je vous conseillerez de ne pas démonter le robot pour récuperer des pièces mais de l'utiliser pour faire des tests et se familiariser avec. Nous n'avons pas eu la chance d'avoir à notre disposition un robot opérationnel en début d'année car le robot précédent ne marchait pas du tout. Utilisez ce robot pour faire des test, vous verrez beaucoup plus rapidement les améliorations à apporter ainsi que les choses à faire ou à ne pas faire…

Surtout, n'hésiter pas à demander des conseils à notre équipe.

Pour me contacter :

Nicolas Bigand
nicolas.bigand@voilà.fr
tel : 06.72.26.31.52